ANÁLISIS COMPARATIVO DE ALGORITMOS PARA OBTENER TRIANGULACIONES DELAUNAY

Vénere Marcelo Javier
Dari Enzo Alberto
División Mecanica Computacional
Centro Atómico Bariloche

RESUMEN

Para realizar triangulaciones Delaunay, tanto en 2-D como en 3-D, el método más utilizado es el algoritmo incremental de Watson. En este trabajo analizamos en detalle diferentes implementaciones del mismo, prestando especial atención a sus costos computacionales (tiempo de cálculo, y memoria requerida). Se presenta además un algoritmo para la regeneración del borde en 2-D que puede resolver cualquier situación, y produce elementos de la mejor calidad posible.

ABSTRACT

The best way to obtain 2-D or 3-D Delaunay triangulation is to use the Watson incremental algorithm. In the current work we perform a detailed analysis of different implementations of it, paying special atention to the robustness, and computational cost.

We also present an algorithm for the boundary regeneration in 2-D, that can solve any situation and produce elements with the best posible quality.

INTRODUCCIÓN

Los algoritmos más utilizados para producir triangulaciones Delaunay [1], están basados en la propuesta realizada simultáneamente por Bowyer [2] y Watson [3] en 1981. La idea básica es válida para cualquier número de dimensiones, y consiste en incorporar secuencialmente los nodos a una triangulación Delaunay previa, de tal forma que la nueva triangulación siga siendo Delaunay. Esto es relativamente simple gracias a dos propiedades de estas triangulaciones:

Test Delaunay: La esfera circunscripta a un símplice no contiene a ningún otro punto de la triangulación.

Propiedad 1 : Los símplices que fallan el test Delaunay con un punto dato que no pertenece a la triangulación, forman un conjunto simplemente conexo.

Propiedad 2 : La triangulación que se obtiene al eliminar estos símplices y agregar aquellos formados por las caras de la cavidad resultante y el punto dato, es también Delaunay.

Desde el punto de vista de generación de redes, este algoritmo es sólo una de las etapas del proceso total, ya que el mismo requiere que ya estén definidos los puntos que formarán la red, y necesita además una triangulación Delaunay que englobe a todos esos puntos. Normalmente los datos de que se dispone son los contornos de la geometría que se quiere triangular, que en los casos de interés, puede ser no-convexa e incluso multiplemente conexa; y un diámetro promedio de los elementos, que permite definir el grado de densificación que va a tener la red. Por lo tanto el proceso total requiere de las siguientes etapas:

1- DISCRETIZACIÓN DEL CONTORNO, Y TEST
2- GENERACIÓN DE NODOS INTERIORES
3- GENERACIÓN DE UNA RED ENGLOBANTE
4- INCORPORACIÓN SECUENCIAL DE LOS NODOS
5- REGENERACIÓN DEL CONTORNO
6- BORRADO DE LOS ELEMENTOS EXTERIORES

1- Discretización del contorno: En esta etapa se generan las aristas sobre todos los contornos (uno exterior, y puede haber varios interiores). Este trabajo conviene realizarlo en forma semi-interactiva: En primer lugar se va recorriendo el contorno y generando los puntos a intervalos igual al diámetro promedio de los elementos; y posteriormente puede ser necesario que el usuario reacomode algunos puntos para que queden en lugares particulares del contorno. En todos los casos va a ser necesario verificar que no se produzcan cortes entre aristas (ver fig. 1); si esta situación se produce, hay dos posibles soluciones: Cambiar el diámetro promedio o desplazar en forma interactiva, los puntos que forman esas aristas.

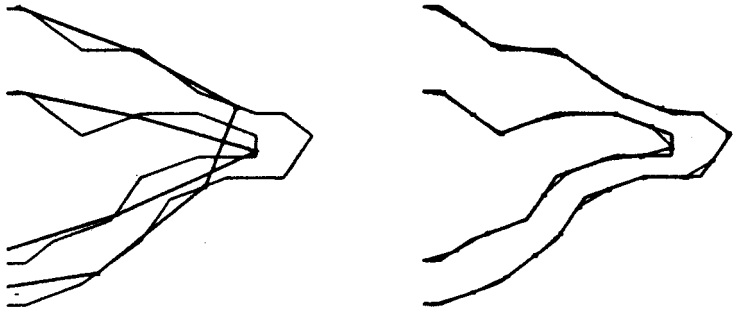


Fig. 1: Intersección de contornos

2- Generación de los nodos interiores: Consiste en definir las coordenadas de los nodos interiores a la red. Si bien tanto la implementación como el costo computacional de esta etapa no presenta problemas, resulta importante utilizar un buen algoritmo ya que una mala distribución de los nodos va a conducir necesariamente a elementos de mala calidad. Existen varias propuestas en la literatura [4, 5, 6] para generar estos nodos. Quizás una de las más apropiadas es la utilización de un árbol (quadtree en 2-D o octree en 3-D) [6], que incluso permite realizar densificaciones locales en forma natural. En nuestro caso hemos utilizado la propuesta de Lo [4], en la que los nodos son colocados sobre rectas horizontales espaciadas una distancia igual al diámetro promedio. Su única desventaja es no permitir densificaciones locales; para ello utilizamos otras técnicas [7, 8, 9] en una etapa posterior.

3- Generación de una red englobante: Esta etapa es sumamente sencilla y consiste en definir una red Delaunay que englobe a todos los puntos generados. Típicamente se toman dos triángulos en 2-D o cinco tetraedros en 3-D (Fig. 2).

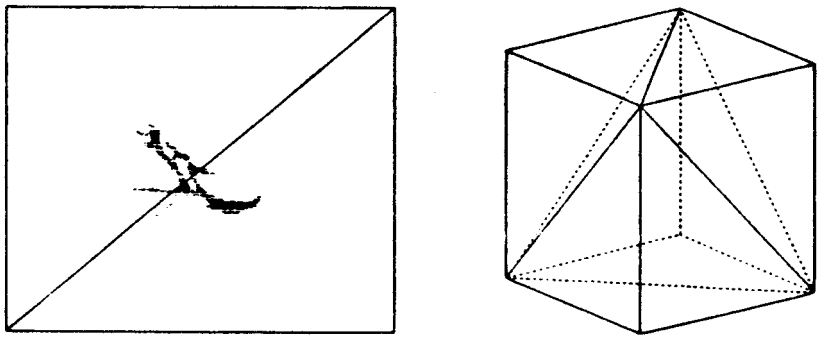


Fig. 2: Redes englobantes en 2-D y 3-D

4- **Incorporación secuencial de los nodos:** Es en esta etapa que se utiliza el algoritmo incremental de Watson, siendo la más costosa de todo el proceso de generación. En la siguiente sección analizaremos en detalle diferentes implementaciones de la misma.

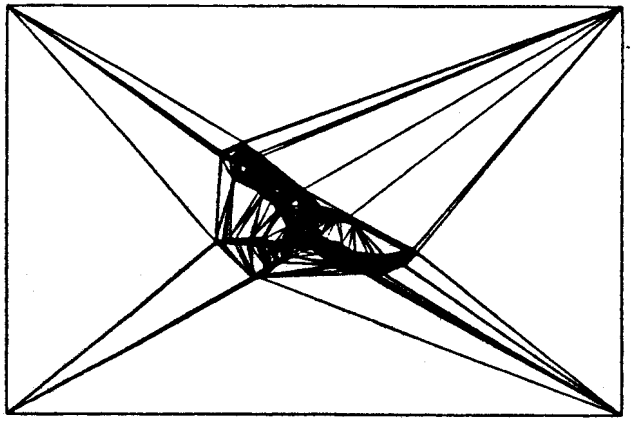


Fig. 3: Red generada por el algoritmo de Watson.

5- **Regeneración del contorno:** En general es posible que aristas que formaban el contorno original, no existan en la triangulación obtenida al aplicar el algoritmo de Watson (fig. 9). En estos casos va a ser necesario modificar localmente la triangulación de forma de crear nuevamente estas aristas. Son también varias las propuestas que aparecen en la literatura para solucionar este problema [10, 11], aunque algunas de ellas no pueden resolver todas las situaciones. Más adelante presentamos un método totalmente robusto para el caso de 2-D, con la atractiva cualidad de producir los elementos de la mejor calidad posible.

6- **Borrado del exterior:** A esta altura se dispone de una triangulación del recinto de interés, pero hemos generado tambien varios triángulos que no pertenecen al mismo (fig. 3), y deben ser removidos. Existen varias posibilidades para realizar esta tarea (en forma automática se sobre-entiende); y sin duda la más económica es la de "llenado de un recinto": Dado un elemento en el exterior, este es eliminado, y se marcan para eliminar a todos sus vecinos siempre que no se cruce el contorno dato. Este algoritmo recursivo requerirá entonces un elemento marcado por cada recinto exterior no conectado (dos en el caso de la fig. 4).

MÉTODO INCREMENTAL DE WATSON

En esta sección describiremos y compararemos cuatro algoritmos posibles para implementar este método. El primero (algoritmo A) es

Fig. 4: Red final despues de borrar los elementos exteriores.

la implementación más obvia de la idea propuesta por Watson, y se puede esquematizar de la siguiente forma:

```
Para cada punto P a incorporar
   Para cada elemento E ya generado
      E falla el test Delaunay con P ? => Seleccionar a E
   Próximo elemento E
   Buscar el contorno del conjunto de elementos seleccionados
   Generar nuevos elementos uniendo cada arista del contorno con P
   Calcular centros y radios de los círculos circunscriptos de los
                                                        nuevos elementos
Próximo punto P
```

Este algoritmo presenta dos serios problemas: en primer lugar los errores de redondeo pueden conducir a selecionar un conjunto de elementos inválido (ver fig. 5); y en segundo lugar el costo computacional del mismo crece como O(N**2).

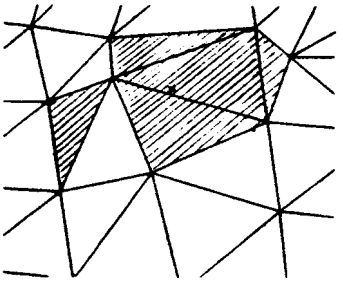


Fig. 5: Selección de elementos inválida

Una posibilidad para solucionar el primer problema, es seleccionar sólo aquellos elementos que además de fallar el test Delaunay, estén conectados (por una arista en 2-D o una cara en 3-D) al conjunto ya seleccionado. De esta forma se garantiza que siempre el conjunto de elementos seleccionados será válido, y por lo tanto también lo será la nueva triangulación. Para ello, en primer lugar se busca al elemento que contiene al nuevo punto, se lo selecciona, y se

marcan para analizar a sus vecinos (tres en 2-D, cuatro en 3-D). Este proceso se repite para todos los elementos marcados que fallan el test Delaunay.

Se encontró que efectivamente este es un algoritmo robusto ya que no presentó problemas en todos los casos analizados (2-D) aún trabajando en simple precisión y con redes de más de 10000 nodos. Sin embargo el problema del costo O(N**2) sigue estando presente, ya que para cada nuevo punto que se incorpora es necesario analizar todos los elementos generados hasta encontrar al que lo contiene.

De todas formas para el caso de 2-D el costo O(N**2) puede resultar molesto, pero no un serio problema ya que N estará en el orden de 1000 a 10000, lo que lleva a un tiempo de generación de varios segundos a algunos minutos en una μVAX II (0.15 Mflops) (ver fig. 7). Sin embargo en 3-D, N puede llegar a 10**5 e incluso a 10**6 lo que lleva a tiempos de cientos de días en una μVAX II; y aunque se disponga de algún procesador más veloz, los tiempos siguen siendo inaceptables.

Para determinar si el punto está dentro de un determinado elemento se debe verificar que el mismo esté del lado "interior" de las tres aristas (cuatro caras en 3-D), es decir que las áreas de los triángulos 12P, 23P, 31P en la fig. 6 sean positivas. Si una de estas áreas es negativa significa que el punto está del otro lado de esa arista, por lo que una mejora muy sencilla del algoritmo anterior sería analizar el elemento vecino en esa dirección. De esta forma no será necesario analizar todos los elementos ya generados, sino solamente los que están sobre "la línea" que va desde el elemento de partida (cualquiera) hasta el que contiene al punto. De esta manera, el orden será O(N**1.5) en 2-D y O(N**1.333) en 3-D. El costo de este algoritmo se puede ver en la fig. 7 (algoritmo B).

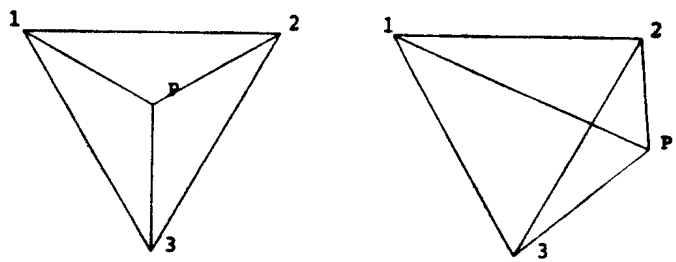


Fig. 6: Punto dentro o fuera de un triángulo.

Para poder disminuir este orden aún más es necesario recurrir a una clasificación de los elementos ya generados, de forma de analizar solamente un grupo reducido y constante de elementos para cada nuevo punto que se incorpora. Una posibilidad es utilizar un árbol (quadtrees en 2-D o octrees en 3-D), que en cada hoja contenga los elementos que tienen intersección no nula con la misma. De esta forma dado un nuevo punto, se ubica en que hoja cae (costo O(LogN)) y luego se analizan sólo los elementos allí contenidos. Este algoritmo presenta serias dificultades de implementación ya que el árbol debe

ser continuamente actualizado con los elementos que desaparecen, y los nuevos que se crean; y tiene importantes requerimientos de memoria. En la fig. 7 (algoritmo C) puede verse además que si bien el costo crece con orden menor que O(N**2), el mismo está lejos de competir con el algoritmo B.

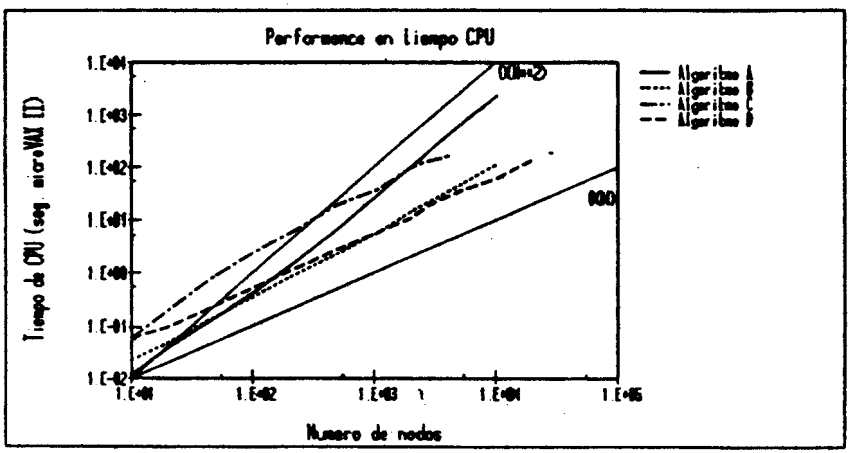


Fig. 7: Costo computacional de distintos algoritmos para el método de Watson.

Una idea más efectiva que la anterior fue propuesta por Baker [11], y consiste en clasificar en un árbol los nodos ya incorporados a la red, y disponer además de una lista que a cada uno de estos nodos le asocie un elemento unido a él. De esta forma al incorporar un nuevo nodo, se busca en que hoja del árbol cae, si la misma no contiene ningún nodo, se busca en las hojas vecinas hasta encontrar una no vacía. El elemento asociado al nodo de la hoja no vacía puede utilizarse como elemento de partida del algoritmo B. Resulta razonable esperar que el número de elementos que se deberán analizar hasta encontrar al que contiene al punto será muy bajo e independiente del tamaño de la red. Por otro lado, el requerimiento de memoria de este árbol es muy inferior al del algoritmo C, y la actualización del mismo es más sencilla ya que sólo debe agregarse el nuevo nodo incorporado y asociarle alguno de los elementos recientemente generados. En la fig. 7 (algoritmo D) puede verse que efectivamente el costo es inferior al de los demás, y que ya para redes de más de 1000 nodos resulta el más económico.

ORDEN DE INCORPORACIÓN DE LOS NODOS

En la sección anterior no hemos hecho ninguna consideración sobre el orden en que son incorporados los puntos, y, como se verá, aunque no afecta la red final que se obtiene (dado el conjunto de nodos, la triangulación Dealunay es única), juega un papel muy importante en el costo de los algoritmos.

Básicamente hay dos variables que pueden controlarse con el orden de incorporación de los nodos: Número de elementos que fallan el test Delaunay al incluir un nuevo punto, y en el caso del algoritmo B, el costo de la búsqueda de estos elementos.

El costo O(N**1.5) del algoritmo B proviene de que en la etapa de busqueda, al partir de un elemento cualquiera, se deberan analizar O(N**0.5) elementos hasta encontrar al que contiene al punto. Si la numeración de los nodos (orden en que son incorporados), fuera de tal manera que nodos cercanos entre sí tengan también numeración cercana, y si se toma como elemento de partida al último generado es de esperar que la búsqueda involucre a unos pocos elementos y sea independiente de N. En la fig. 8 puede verse que efectivamente esto es lo que ocurre. Lamentablemente en la misma figura puede observarse que con esta numeración el número de elementos que fallan el test Delaunay aumenta aproximadamente como O(N**0.5), con lo que el orden del algoritmo sigue siendo O(N**1.5).

Otra dificultad de esta numeración es que durante el proceso de generación continuamente se tienen elementos tipo agujas, lo que aumenta considerablemente el error de redondeo, y limita el número de nodos con que se puede trabajar (inferior a 10000 en simple precisión).

Nótese que con esta numeración cualquier algoritmo va a tener un orden mayor o igual a O(N**1.5). Por ello si queremos explotar las posibilidades del algoritmo propuesto por Baker, será necesario renumerar previamente los nodos en forma adecuada. Es posible hallar una numeración de tal manera que cada nuevo punto se inserte en una

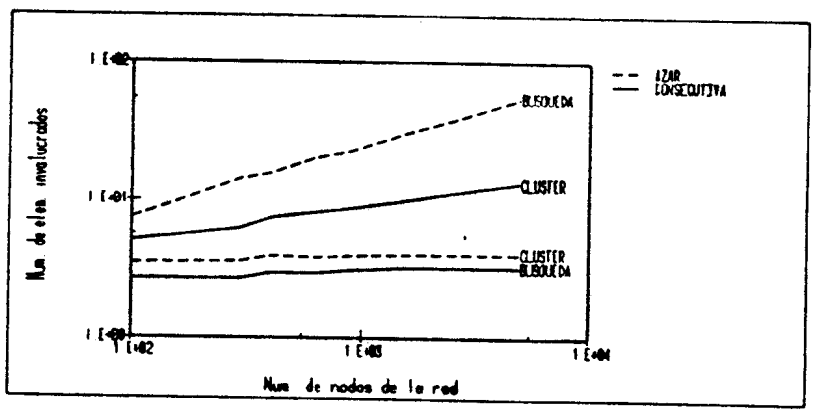


Fig 8: Número de elementos involucrados en la búsqueda y cluster para diferentes numeraciones

red cuasi-regular. La idea es evitar la formación (durante la generación) de elementos con grandes círculos circunscriptos, que hacen aumentar el tamaño del cluster y el error de redondeo. Una posibilidad muy simple es tomar una numeración al azar, en la fig. 8

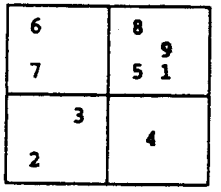
puede verse que en este caso efectivamente el número de elementos se mantiene casi constante, independientemente del tamaño de la red. En nuestro caso hemos utilizado un algoritmo que garantiza la cuasi-regularidad de la red en todo momento de la generación, que se puede sintetizar de la siguiente manera:

Paso 1: Armar un árbol (quadtree o octree) de forma que cada hoja tenga a lo sumo un nodo.

Paso 2: Marcar todas las ramas como no-procesadas y las hojas como "ramas" procesadas, es decir que tienen una lista asociada de nodos ya ordenados (inicialmente 1 nodo o ninguno).

Paso 3: Para todas las ramas no-procesadas, cuyos 4 (8) hijos sí lo estén: Crear una lista tomando un nodo de la lista de cada hijo, hasta agotarlas. Marcar la rama como ya procesada. Si la rama no es la raíz, ir a Paso 3.

Paso 4: Se toma como orden de incorporación de los nodos, la lista asociada a la raíz.



HIJO	LISTA
1	6,7
2	8,5,9,1
3	3,2
4	4

LISTA FORMADA: 6,8,3,4,7,5,2,9,1

Fig. 9: Paso 3 de renumeración para minimizar el cluster.

REGENERACIÓN DEL CONTORNO

Ya hemos mencionado que luego de incorporar todos los nodos a la triangulación, es posible que en ésta no aparezcan algunas aristas del contorno original. En este caso, antes de borrar los elementos exteriores se deberán regenerar estas aristas.

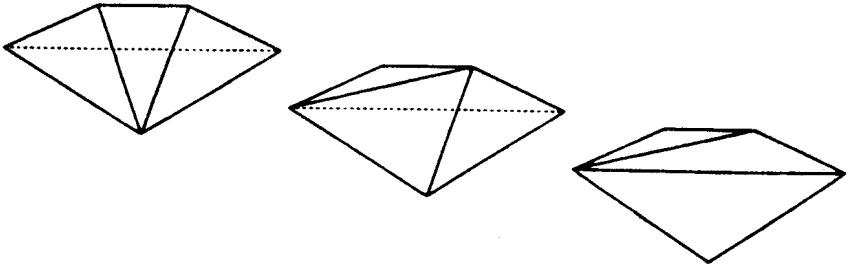


Fig. 10: Regeneración de una arista del contorno

En primer lugar es necesario buscar los elementos que son intersectados por la arista faltante (fig. 10). Luego se podrá modificar localmente la triangulación de forma de que aquella aparezca. Probablemente el algoritmo más utilizado para esta tarea está basado en cambiar las diagonales entre pares de triángulos hasta generar la arista deseada (fig. 10). La versión en 3-D de este algoritmo fue propuesta por George [10] y consiste en tomar los tetraedros de a dos, crear una arista y borrar una cara formando de esta forma tres tetraedros en lugar de dos (fig. 11).

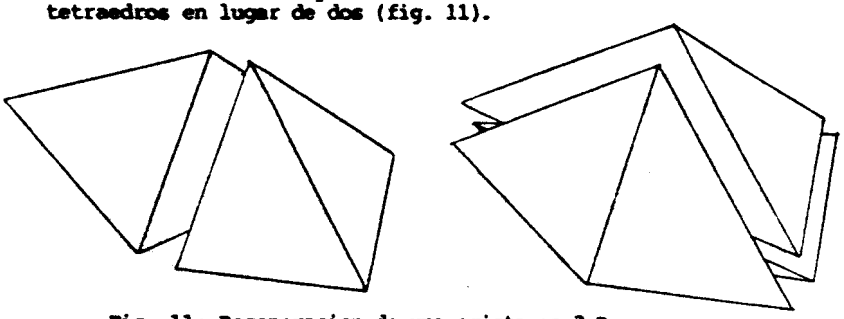


Fig. 11: Regeneracion de una arista en 3-D.

Lamentablemente este algoritmo relativamente sencillo no es totalmente robusto, ya que hay situaciones que no puede resolver como se muestra en la fig. 12. Para subsanar este problema es necesario recurrir a una idea bastante más compleja que se puede sintetizar de la siguiente forma:

1- Dado el conjunto de elementos intersectados por la arista faltante, determinar el contorno del mismo. De esta forma quedan definidos dos polígonos separados por la arista faltante (fig. 13)

2- Para cada uno de estos polígonos se procede de la siguiente manera:
   a) Se busca el ángulo más agudo formado por dos aristas del mismo.
   b) Se verifica que el triángulo formado por estas dos aristas y una nueva que se crearía opuesta al ángulo no contenga ningún nodo del polígono, en este caso se forma el nuevo triángulo. En caso contrario se toma el siguiente ángulo más agudo y se repite la verificación. Es sencillo ver que siempre existirá un par de aristas del polígono con las que se podrá formar un triángulo.
   c) Se eliminan del polígono original las dos aristas reemplazándolas por la nueva.
   d) Si el polígono resultante no es un triángulo volver al punto a) En caso contrario se forma el último elemento.

Analizando con detenimiento esta idea, puede verse que puede resolver cualquier situación. Su costo es sin ninguna duda muy superior al del algoritmo anterior, pero ésto no es una dificultad ya que de todas maneras, el orden va a ser el mismo (O(N**0.5)), con lo que rápidamente será irrelevante dentro del proceso total de generación (ver fig. 15).

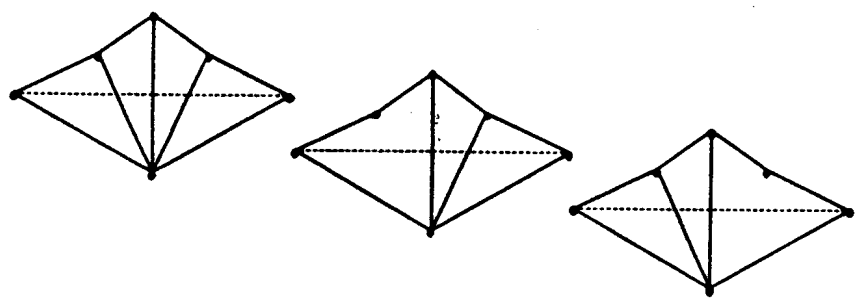


Fig. 12: Caso patológico de regeneración de una arista

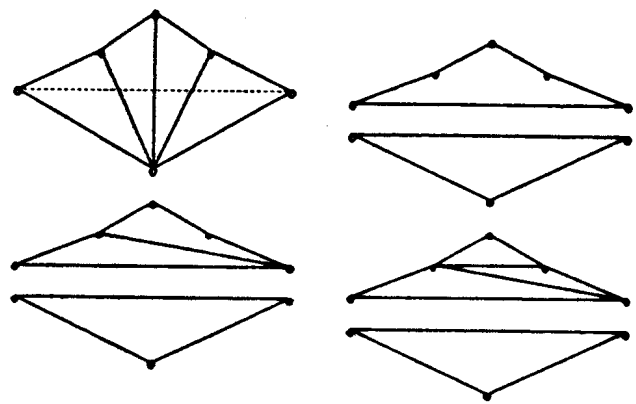


Fig. 13: Algoritmo alternativo para regenerar el contorno.

CONSIDERACIONES FINALES

Hemos descripto con cierto detalle las distintas etapas a seguir para generar triangulaciones Delaunay, haciendo especial énfasis en la elección de algoritmos robustos. De esta forma es posible obtener un método casi totalmente automático donde el usuario sólo debe preocuparse por definir adecuadamente el contorno y elegir un tamaño promedio de elemento, obteniendo una red válida con elementos de muy buena calidad.

Con este esquema, un problema que queda sin solución son las densificaciones locales. Para ello utilizamos las tecnicas de partición de elementos [7, 8] ya sea especificando interactivamente los elementos de las zonas que se desean densificar, o en forma adaptiva si se dispone de una solución. En particular en el caso de 3-D entendemos que las técnicas adaptivas resultarán imprescindibles debido a las dificultades que presenta el trabajo interactivo [9]. En la figura 14 se muestra un ejemplo en que posteriormente a la triangulación Delaunay se densificó en forma interactiva.

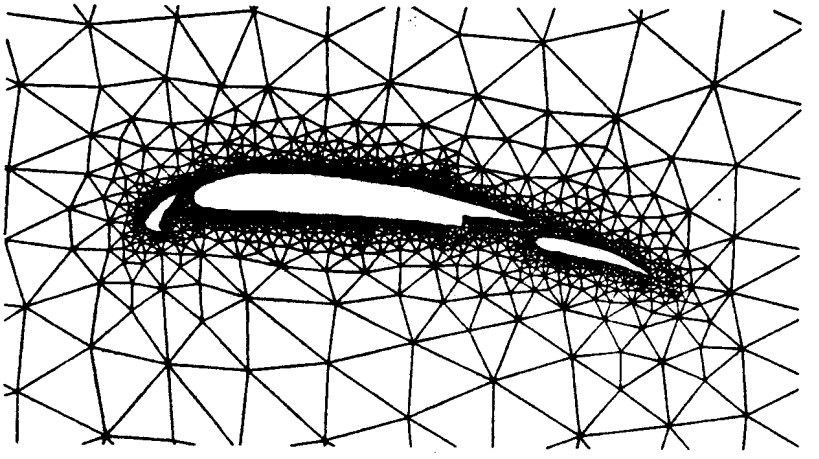


Fig. 14: Triangulación del un perfil alar con densificaciones locales

Finalmente en la figura 15 se muestran los costos de las distintas etapas del método de triangulación descripto, donde se observa que casi la totalidad del tiempo de cálculo se emplea en el algoritmo incremental de Watson, y que prácticamente el mismo crece con orden O(N) (el termino LogN resulta inapreciable).

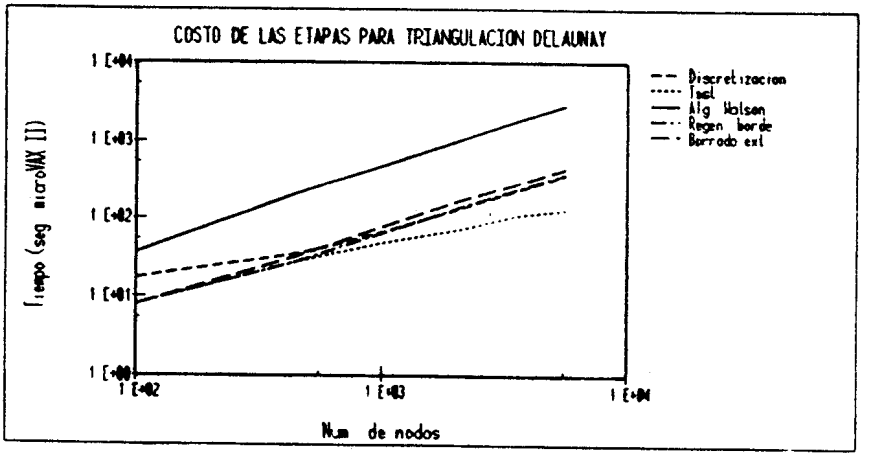


Fig. 15: Costos de las distintas etapas del algoritmo completo.

BIBLIOGRAFÍA

[1] Delaunay, B., "Sur la Sphere vide", Bull. Acad. Science USSR VII: Class. Scil, Mat. Nat. pp 793-800, 1934.

[2] Bowyer, A., "Computing Dirichlet Tessellations", The Computer Journal, Vol. 24, N° 2, pp 162-166, 1981.

[3] Watson, D.F., " Computing de n-Dimensional Delaunay Tessellation with Application to Voronoi Polytopes", The Computer Journal, Vol. 24, N° 2, pp 167-172, 1981.

[4] Lo, S.H., "A new mesh generation scheme for arbitrary planar domains", Int. Journal Num. Meth. in Eng. 21, pp 1403-1426, 1985

[5] Frey, W, "Selective Refinement: A New Strategy for Automatic Node Placement in Graded Triangular Meshes", Int. Journal Num. Meth. in Eng. 24, pp 2183-2200, 1987.

[6] Schroeder W., Shephard M., "An O(N) Algorithm to Automatically Generate Geometric Triangulations Satidfying the Delaunay Criteria", Engineering with Computers 5, pp 177-193, 1989.

[7] Rivara, M.C., "Algorithms for refining triangular grids suitable for adaptive and multigrid techniques", Int. Journal Num. Meth. in Eng. 21, pp 745-756, 1984.

[8] Rivara, M.C., "Mesh Refinement processes based on the generalised bisection of simplices", SIAM J of Num. Ana. 21, pp 604-613, 1984

[9] Venere M.J., "Técnicas Adaptivas en Cálculo Numérico para Problemas en Dos y Tres Dimensiones", Proceedings de la Segunda Conferencia Franco-Chilena en Matemáticas Aplicadas.

[10] George P.L., Hecht F., Saltel E., "Constraint on the Boundary and Automatic Mesh Generation", Proceedings of Second International Conference on Numerical Grid Generation in Computational Fluid Mechanics, Miami. pp 589-597.

[11] Baker T.J., "Automatic Mesh Generation for Complex Three-Dimensional Regions Using a Constrained Delaunay Triangulation", Engineering with Computers, Vol 5, pp 161-175, 1989.